**Name: Srirupa Guha**

**Date**: September 1, 2020

**Course**: IT FDN 110: Introduction to Programming (Python)

**Assignment**: 8

**Title**: Classes, Attributes and Methods

**Introduction**:

This week we learned about objects and object-oriented programming, classes, various properties of classes that are class specific (fields) and those that are object specific (attributes), methods and decorators.

**Objects and Object-Oriented Programming:**

Object-Oriented programming or OOP is different from the traditional procedural programming in the sense that data and code are organized in objects and the objects interact with each other. Objects have both data as attributes, as well as methods that are applicable to the objects. So instead of procedural programming with lists and list manipulation, it is easier to have objects interact via methods with the “outside code”, or code that is not part of the object.

**Classes**

Classes are blueprints for objects. When a new object is created from a class, it is called instantiation. Classes are blueprints for objects, like a blueprint for houses in neighborhoods. For example, there can be a car class with properties like make, model, color, mileage, type, and various objects can be instantiated from this car class, such as a white Honda Civic sedan with 50,000 miles, or a burgundy Subaru Forrester hatchback with 20,000 miles, etc. Classes can be as simple as: class car(), or more complex with class-specific fields, constructor, attributes, properties and methods, such as:

class car():

#----constructor-----

def \_\_init\_\_(self, make, model, color, mileage):

self.\_\_make = make

self.\_\_model = model

self.\_\_color = color

self.\_\_mileage = mileage

#-----Properties-----

def get\_make(self);

return self.\_\_make

def get\_model(self);

return self.\_\_model

def get\_color(color);

return self.\_\_color

def get\_mileage(self);

return self.\_\_mileage

**Attributes**

Attributes are data about the unique object. They are not properties of the class that is common to all the objects – they are properties that belong to only the object. Example – different cars have different makes, models, colors, mileage, etc. Attributes usually are listed just under the constructor in the class. Constructors are used to easily instantiate new objects. You can restrict direct access to an object’s attributes by adding the double underscore to the attribute name.

**Methods**

Methods are code in the object used for handling the attributes of an object. In cases were an attribute(s) is/are private, the only way for outside code to gain access or to modify these private attributes are by the methods of the object. Methods can also validate the attributes that are set, such as raising exceptions if the data is not of a particular type (like non-numeric data for mileage, for example). Methods execute a lot of code that would have been present in long, procedural programs with subroutines.

**Documentation**

Docstrings are a good way of documenting classes, and explaining the various properties and methods of classes. Further, class methods can also benefit from docstring documentation, especially if the code in the method is elaborate. This is so that some other software developer can understand the purpose of the code and debug it easily.

A very good website that explained object-oriented programming is: [Realpython Object Oriented Programming](https://realpython.com/python3-object-oriented-programming/) [[1]](#footnote-1)

Ultimately, I put all this knowledge to test and wrote and executed my Python program:

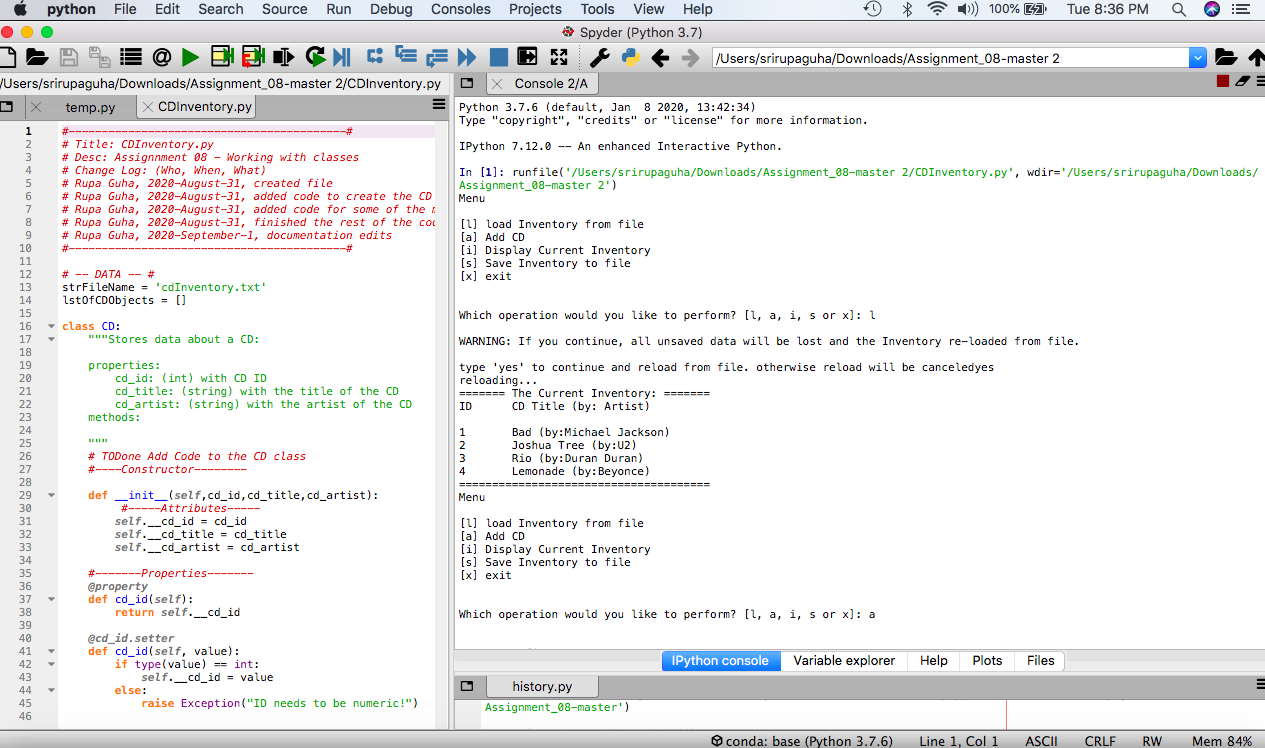


Figure Page 1 of Program in Test

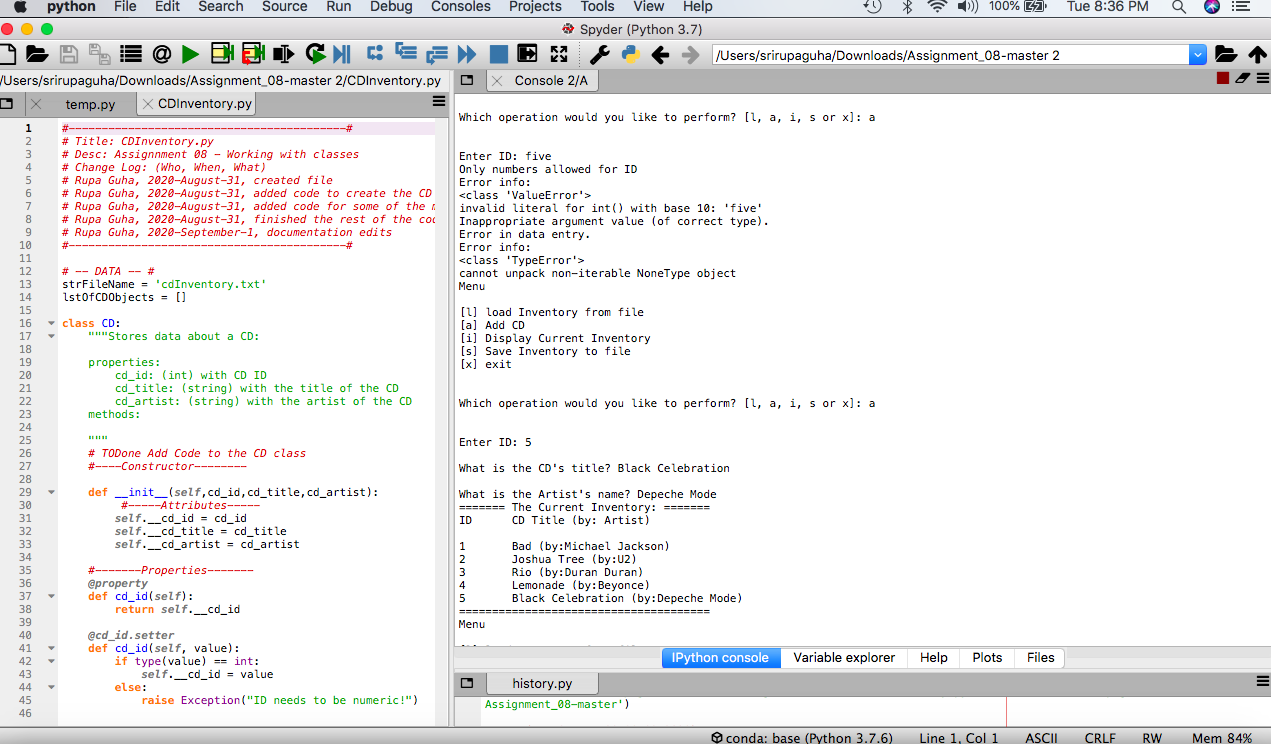


Figure Page 2 of Program in Test

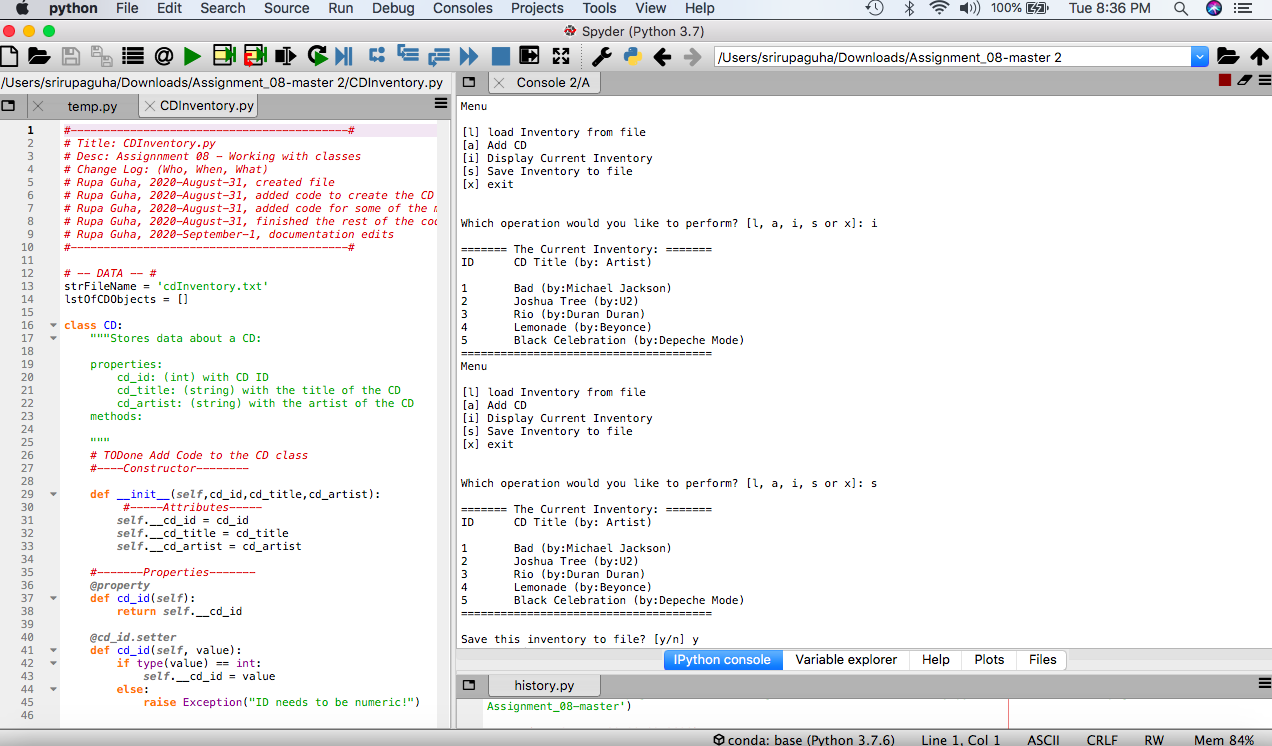


Figure page 3 of Program in Test

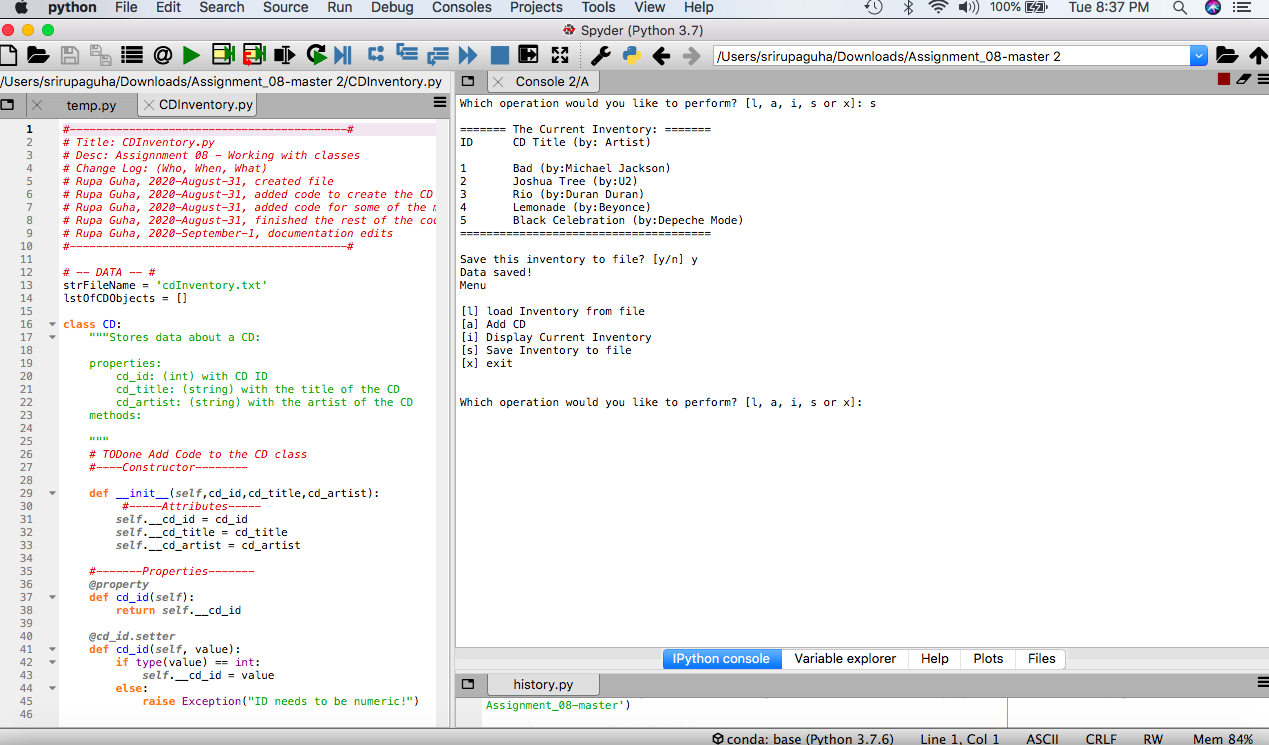


Figure Final Page of Program in Test

**Summary**

Finally, this week we learned objects, classes, various properties of classes that are class specific (fields) and those that are object specific (attributes), methods and decorators.

**Appendix**

The Python assignment code for this week:

1. #------------------------------------------#
2. # Title: CDInventory.py
3. # Desc: Assignnment 08 - Working with classes
4. # Change Log: (Who, When, What)
5. # Rupa Guha, 2020-August-31, created file
6. # Rupa Guha, 2020-August-31, added code to create the CD class
7. # Rupa Guha, 2020-August-31, added code for some of the methods in FileIO and IO classes as well as main body
8. # Rupa Guha, 2020-August-31, finished the rest of the code - IO class methods (add routines) and main body
9. # Rupa Guha, 2020-September-1, documentation edits
10. #------------------------------------------#
12. # -- DATA -- #
13. strFileName = 'cdInventory.txt'
14. lstOfCDObjects = []
16. **class** CD:
17. """Stores data about a CD:
19. properties:
20. cd\_id: (int) with CD ID
21. cd\_title: (string) with the title of the CD
22. cd\_artist: (string) with the artist of the CD
23. methods:
25. """
26. # TODone Add Code to the CD class
27. #----Constructor--------
29. **def** \_\_init\_\_(self,cd\_id,cd\_title,cd\_artist):
30. #-----Attributes-----
31. self.\_\_cd\_id = cd\_id
32. self.\_\_cd\_title = cd\_title
33. self.\_\_cd\_artist = cd\_artist
35. #-------Properties-------
36. @property
37. **def** cd\_id(self):
38. **return** self.\_\_cd\_id
40. @cd\_id.setter
41. **def** cd\_id(self, value):
42. **if** type(value) == int:
43. self.\_\_cd\_id = value
44. **else**:
45. **raise** Exception("ID needs to be numeric!")

48. @property
49. **def** cd\_title(self):
50. **return** self.\_\_cd\_title
52. @cd\_title.setter
53. **def** cd\_title(self, value):
54. **if** type(value) == str:
55. self.\_\_cd\_title = value
56. **else**:
57. **raise** Exception("Title needs to be a string!")
59. @property
60. **def** cd\_artist(self):
61. **return** self.\_\_cd\_artist
63. @cd\_artist.setter
64. **def** cd\_artist(self, value):
65. **if** type(value) == str:
66. self.\_\_cd\_artist = value
67. **else**:
68. **raise** Exception("Artist needs to be a string!")



73. # -- PROCESSING -- #
74. **class** FileIO:
75. """Processes data to and from file:
77. properties:
79. methods:
80. save\_inventory(file\_name, lst\_Inventory): -> None
81. load\_inventory(file\_name): -> (a list of CD objects)
83. """
84. # TODone Add code to process data from a file
85. @staticmethod
86. **def** load\_inventory(file\_name, lstOfCDs):
87. """ Loads inventory data from the text file into a list in object format."""
88. objFile = None
89. lstOfCDs.clear()  # clearning any existing data
90. objFile = open(file\_name, 'r')
92. **try**:
93. **for** line **in** objFile:
94. data = line.strip().split(',')
95. cdObj = CD(data[0],data[1],data[2])
96. lstOfCDs.append(cdObj)
97. **except**:
98. ("Something happened here - maybe there is nothing in the file yet?")
99. objFile.close()

102. # TODone Add code to process data to a file
103. @staticmethod
104. **def** save\_inventory(file\_name, lst\_Inventory):
105. """Saves data from a list of objects to an inventory file"""
106. new\_line = ""
107. objFile = None
108. counter = 0
110. **while** counter < len(lst\_Inventory):
111. str\_cd\_id = str(lst\_Inventory[counter].cd\_id)
112. new\_line = new\_line + str\_cd\_id + ","
113. new\_line = new\_line + lst\_Inventory[counter].cd\_title + ","
114. new\_line = new\_line + lst\_Inventory[counter].cd\_artist + ","
115. new\_line = new\_line[:-1] + "\n"
116. counter += 1
118. objFile = open(strFileName, "w")
119. objFile.write(new\_line)
120. objFile.close()
121. **print**("Data saved!")

124. # -- PRESENTATION (Input/Output) -- #
125. **class** IO:
126. # TODone add docstring
127. """Displays data to the screen:
129. properties: None
131. methods:
132. print\_menu()
133. menu\_choice()
134. show\_inventory()
135. ask\_user\_data()
137. """
139. # TODone add code to show menu to user
140. @staticmethod
141. **def** print\_menu():
142. """Displays a menu of choices to the user
144. Args:
145. None.
147. Returns:
148. None.
149. """
151. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
152. **print**('[s] Save Inventory to file\n[x] exit\n')
154. # TODone add code to captures user's choice
155. @staticmethod
156. **def** menu\_choice():
157. """Gets user input for menu selection
159. Args: None.
161. Returns: choice (string): a lower case sting of the users input out of the choices l, a, i, s or x
163. """
164. choice = ' '
165. **while** choice **not** **in** ['l', 'a', 'i', 's', 'x']:
166. choice = input('Which operation would you like to perform? [l, a, i, s or x]: ').lower().strip()
167. **print**()  # Add extra space for layout
168. **return** choice
170. # TODone add code to display the current data on screen
171. @staticmethod
172. **def** show\_inventory(lstObj):
173. """ Prepares the CD inventory for proper display and formatting to the screen
175. Args: the inventory table of CD objects
176. Returns: None
178. """
180. counter = 0
182. **print**('======= The Current Inventory: =======')
183. **print**('ID\tCD Title (by: Artist)\n')
185. **while** counter < len(lstObj):
186. **print**('{}\t{} (by:{})'.format(lstObj[counter].cd\_id, lstObj[counter].cd\_title, lstObj[counter].cd\_artist))
187. counter += 1
188. **print**('======================================')
190. # TODone add code to get CD data from user
191. @staticmethod
192. **def** ask\_user\_data():
193. """Asks for user data - the ID of the new CD, the Title and the Artist
195. Args: None
196. Returns: The ID, the CD Title and the Artist of the title
197. """
199. # catching errors like entering non-numeric entries
200. **try**:
201. ID = int(input('Enter ID: ').strip())
202. Title = input('What is the CD\'s title? ').strip()
203. Artist = input('What is the Artist\'s name? ').strip()
204. **return** ID, Title, Artist
205. **except** ValueError as e:
206. **print**("Only numbers allowed for ID")
207. **print**("Error info: ")
208. **print**(type(e),e,e.\_\_doc\_\_, sep="\n")

211. # -- Main Body of Script -- #
213. # TODone Add Code to the main body
215. # Load data from file into a list of CD objects on script start
216. FileIO.load\_inventory(strFileName, lstOfCDObjects)
218. **while** True:
219. # Display menu to user
220. IO.print\_menu()
221. strChoice = IO.menu\_choice()
223. # let user exit program
224. **if** strChoice == 'x':
225. **break**   # end the program
227. # let user load inventory from file
228. **if** strChoice == 'l':
229. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
230. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled')
231. **if** strYesNo.lower() == 'yes':
232. **print**('reloading...')
233. FileIO.load\_inventory(strFileName, lstOfCDObjects)
234. IO.show\_inventory(lstOfCDObjects)
235. **else**:
236. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
237. IO.show\_inventory(lstOfCDObjects)
238. **continue**    # start loop back at top.
240. # let user add data to the inventory
241. **if** strChoice == 'a':
242. # catching error when erroneous data was not passed from IO.ask\_user\_data()
243. **try**:
244. intID, strTitle, stArtist = IO.ask\_user\_data()
246. **except** TypeError as e:
247. **print**("Error in data entry.")
248. **print**("Error info: ")
249. **print**(type(e),e, sep="\n")
250. **continue**
252. newObj = CD(intID, strTitle, stArtist)  #   creating a new CD object with the new user data
253. lstOfCDObjects.append(newObj)   #   adding this new object to the list of CD oblects
254. IO.show\_inventory(lstOfCDObjects)   #   displaying to the user the current inventory with the new CD
255. **continue**    # start loop back at top.

258. # show user current inventory
259. **if** strChoice == 'i':
260. IO.show\_inventory(lstOfCDObjects)
261. **continue**    # start loop back at top.
263. # let user save inventory to file
264. **if** strChoice == 's':
265. IO.show\_inventory(lstOfCDObjects)
266. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
268. **if** strYesNo == 'y':
269. FileIO.save\_inventory(strFileName, lstOfCDObjects)
270. **else**:
271. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
272. **continue**    # start loop back at top.

1. <https://realpython.com/python3-object-oriented-programming/> Retrieved September 1, 2020 [↑](#footnote-ref-1)